ISRN SICS-T--94/10-SE

When SUC met CLE

Parsing tagged unrestricted text in the Swedish Core Language Engine

by

Nikolaj Lindberg and Lena Santamarta

SICS Technical Report
T94:10
ISSN 1100-3154

T94:10 ISSN 1100-3154

When SUC met CLE

Parsing tagged unrestricted text in the Swedish Core Language Engine

Nikolaj Lindberg nikolaj@sics.se

Lena Santamarta santa@sics.se

November 1994

Natural Language Processing Group
Swedish Institute of Computer Science
Box 1263, S–164 28 KISTA, Sweden

Abstract:
This paper describes a way of feeding part of speech tagged Swedish text to the syntactic parser of the Swedish Core Language Engine, in order to get automatically produced syntactic analyses of unrestricted written text. The idea is to later on manually disambiguate and correct the output of the parser, or, in other words, to start building a treebank — a corpus of syntactically analyzed text.

After describing an already existing treebank for English and presenting the material used, a detailed account of the process of transforming the tagged text to a format suitable for the parser is given.

Keywords: Tree-bank, corpora, computational lexica

Also Published: This report is also available as a Bachelor of Art Thesis in Computational Linguistics, Department of Linguistics, Stockholm University.

Table of Contents

1 Introduction

1.1 What is the meaning of it all?

The aim of this work is to plant the first tiny seeds of a treebank for Swedish, and the programmes our efforts have resulted in so far merely transform tagged Swedish texts into a format suitable for the parser we use. The language material used is taken from a part of speech and morphologically annotated corpus for Swedish, called SUC (see 3.2). For the syntactic analysis, we use the Swedish Core Language Engine, the CLE (see 3.1).

In the next chapter we describe an already existing treebank for English. In chapter three the tools and material we have used are presented, and in chapter four we explain our own work, which has been supervised by prof. Gunnel Källgren at the Department of Linguistics at the University of Stockholm and leader of the SUC project, and Björn Gambäck, researcher at the Swedish Institute of Computer Science (SICS) and responsible for developing the grammar of the Swedish Core Language Engine.

We have not yet started building syntactic trees from the output analyses of the CLE; this remains to be done as the next step of this project.

1.2 A treebank?

A treebank for a natural language is a collection of sentences, all of which are given syntactic structure by syntactic tagging and bracketing. In other words, phrase structure trees are constructed for each sentence in a corpus.

A treebank can be of use in a host of different areas: For evaluating the performance of parsers and (formal) grammars, for different kinds of analyses of language, for building grammars, for training statistical or machine learning systems, for building lexica and for developing various applications, such as question answering systems, information retrieval, speech processing and more.

2 Background: Other trees in the wood

For English, there exist a few syntactically analyzed corpora. The SUSANNE ("Surface and underlying structural analyses of naturalistic English") corpus, released in 1992, for which Geoffrey Sampson at the University of Sussex, England, is responsible, was built from text taken from the Brown Corpus of American English [Sampson 1992]. It comprises about 128 000 words of manually analysed written English. SUSANNE evolved from a previous work, The Lancaster-Leeds Treebank [Garside *et al* 1987], which was never published. The SUSANNE treebank is somewhat smaller than the Penn treebank presented below (2.1), but is more detailed. In SUSANNE, information about verb transitivity and logical structure of clauses is found, which is not the case with the Penn Treebank.

In the seventies, a corpus of spoken and written Swedish with detailed syntactic information was produced [Teleman 1974]. The work was not really computer-aided, but the results were saved on punch-cards and magnetic tape. The Teleman corpus comprises some 345 000 words of hand-annotated text material, taken from spoken Swedish, quality fiction and essays of school-children. The primary goal was to produce raw-material for sociolinguistic research and for comparative syntactic studies (the aim of the syntactic studies was to compare the structure and complexity of the language of less talented writers with that of more accomplished ones, in order to be able to improve the teaching of Swedish).

2.1 The story of the Penn Treebank

In this section, we will present the Penn Treebank, produced at the University of Pennsylvania and built from a part of speech annotated corpus containing various English texts: Scientific abstracts, agricultural bulletins, fiction, news stories, computer manuals, transcribed spoken language, and more [Marcus *et al* 1993]. In 1993, the Penn Treebank contained nearly three million words.

One reason for us to describe the Penn Treebank is that part of it, the so called ATIS sentences, consisting of spoken language of

the Air Travel Information System domain, has been used in one of SICS´s projects, namely for training a speech-to-speech translation system, the SLT, which is based on the English and the Swedish versions of the Core Language Engine [Agnäs *et al* 1994].

2.2 The parser used in the Penn Treebank project

The parser used when building the Penn Treebank is a purely syntactic one, which does not take semantics or context dependent factors into account. It is furthermore a "cowardly" parser, in that it does not produce analyses it is uncertain of. The output is a string of constituents rather than a complete tree structure. The reason for this is that there are several things a purely syntactic parser cannot handle in a satisfying way (PP attachment, relative clauses, adverbial modifiers and the scope of conjunctions, for example [Marcus *et al* 1993, pp. 322-323]). Such things are left to be taken care of by human annotators. An annotator has access to the context - the entire text a sentence occurs in - and can thus make decisions that the parser cannot. The parser is said to have a good grammatical coverage, so the structures it does produce are often accurate.

In line with the above said, the parser produces one analysis only, and never a set of different analyses, and it is assumed that it is easier for a human annotator to "glue together" separate constituents to a tree, than to choose the right analysis from a set of competing analyses [Marcus *et al* 1993, p. 320]. The annotators´ job is to check whether the constituents are correctly analyzed, to correct faulty ones and to put the constituents together to a complete tree. To their help, annotators have a mouse-based interface, and to attach a constituent, they simply move it to the place in the tree where it is supposed to be placed.

To make the job easier, the constituents presented to the annotator have been stripped of as much information as possible, as visually simple constituents obviously are easier to deal with. For example, if a constituent looks like this when it leaves the parser:

```
(NP (IART "a/DT")
      (NBAR (N "boatload/NN"))
            (PP of/PREP
                  (NP (NBAR
                        (NPL "warriors/NNS"))))),
```

this simplified version is presented to the annotator:

```
(NP a boatload
      (PP of
            (NP warriors))).
```

The resulting trees are said to be "skeletal", since they could have held more information than they do. The rationale behind this is that to build a more detailed treebank would be too time consuming and, furthermore, that some information not found explicitly in a parse tree can be extracted fairly simply by analyzing the tree, by looking for combinations of structures. An example given is that *that*-clauses, for which there is no specific tag, can be found by searching for SBAR-constituents with the word *that* or the "zero variant of *that*", O, in initial position.

2.3 The syntactic tagging of the Penn Treebank

As already explained, the annotator is presented with bracketed and labelled structures (of the same format as the example above), which are to be put together to form complete phrase structure trees. The syntactic tag set consists of 13 tags for different syntactic categories (ADJP, ADVP, NP, PP, VP, etc), one tag for marking cases the annotator cannot decide (X), and four tags for different null elements. The annotators do not have to add any extra syntactic tag that is not found in the tag set of the parser. When an annotator cannot decide how a constituent should be tagged, the X tag is used. In a similar way, a special "pseudo-attachment" tag is used to signal that a constituent has more than one possible attachment. The pseudo-attachment tag can be used when it is impossible to resolve a structural ambiguity, for instance. The philosophy behind the X tag and the pseudo-attachment is similar to the one behind the fact

that the parser only produces analyses it is certain of: The annotators shall only produce analyses they are sure of.

Trained Penn Treebank annotators work at an average speed of 750 words an hour, which means that it takes approximately one year to build a treebank of 2.5 million words for five annotators working three hours a day.

3 Tools and material

In this chapter, the system used for syntactic analysis and the text source are presented. The text source, taken from a part-of-speech tagged corpus (see 3.2), is transformed into a Prolog readable format, in order to use it as input to the main programme (see 4). This programme preprocesses the input to make it suitable for parsing in the Swedish Core Language Engine (see 3.1).

3.1 The core language engine (CLE)

The Core Language Engine (CLE) is a multi-purpose natural language processing system, originally for English, developed at SRI International, Cambridge, written entirely in Prolog [Alshawi (ed.)1992]. A Swedish counterpart has been developed by the Swedish Institute of Computer Science (see for example [Gambäck & Rayner 1992] or [Gambäck *et al* 1994]). When we use the name CLE below, we refer to the Swedish version.

The idea is that the Engine should be able to take care of language processing in any computer system dealing with natural language (English, Swedish and French so far). The CLE can be used for analysis as well as generation. Examples of possible uses are data-base query systems for spoken or written language and machine translation.

The CLE comprises several different modules, each handling different aspects of language. The CLE processes one sentence (or utterance) at a time. With the help of morphological rules and a lexicon, the first processing step makes a morphological analysis of each word and decides what part(s) of speech it may belong to. When the morphology has been taken care of, it is time for syntactic analysis. This task is performed by a left to right shift-

reduce parser and a unification-based grammar. For each grammar rule, there is at least one corresponding semantic rule, and the last step in the process is to give the sentence a semantic representation in a logical form like language.

3.1.1 CLE morphology

The CLE splits the input string in tokens (words, in most cases) and each word is given one or more morphological parses - i.e. the words are assigned grammatical categories. This is done by segmenting the words in stems and affixes, and looking up all possible word stems in a lexicon. When a word stem is found in the lexicon, morphological rules are applied to the word and its affix(es) to decide what part(s) of speech it may belong to, and what morphological properties it has (tense, gender, number, definiteness, etc). A (simplified) morphology rule for forming a noun in the plural (**NBAR --> NBAR + PLUR**) might look thus:

```
morph(nbar_nbar_plural,
      [nbar:[agr=plur, synmorphn=M, def=n, ...],
      nbar:[agr=sing, synmorphn=M,def=n, ...],
      'PLURAL':[synmorphn=M]]).
```

Here the first argument (**nbar_nbar_plural**) simply is the name of the rule that says that an **nbar** can be formed by a singular **nbar** plus a plural ending (for example, **aptitbitar --> aptitbit + ar**). The first element of the list following the rule name is the mother category (the left-hand side of the **NBAR --> NBAR + PLUR** rule), while the second and third elements are the daughter nodes (the right-hand side of the same rule). The feature **synmorphn** (syntactic morphological category for nouns) says that for the rule to be applicable, **synmorphn** must be instantiated with the same declension number for both daughters - for the word stem and the ending, in other words - and that this feature value is passed on to the mother.

Possible endings are found in a lexicon, in which the proper **synmorph** values are found. A word with the feature value **synmorphn=2** can form the plural form with the ending **-ar**, which has the same feature value (has the same declension number as the

noun, that is). A sample of possible "ending entries" for some Swedish plural nouns:

```
lex('-or',['PLURAL':[synmorphn=1,lexform='-or']]).
lex('-ar',['PLURAL':[synmorphn=2,lexform='-ar']]).
lex('-er',['PLURAL':[synmorphn=3,lexform='-er']]).
```

The same mechanism, albeit with a little more complex set of rules, is used for verb and adjective morphology.

3.1.2 The CLE parser

The parser now faces all possible morphological analyses of the words in the input string. It uses the so called shift-reduce strategy to put the word parses together to a complete syntactic analysis. A shift-reduce parser is basically a bottom-up parser that makes top-down predictions based on the constituents it has built: By looking at the bottom-up built constituents, it rules out grammar rules not consistent with constituents already parsed. The output of the parser consists of all possible parses (given the grammar and lexicon described below) of the input.

3.1.3 The beauty and the beast: CLE grammar and lexicon

The CLE uses a unification grammar, which is a kind of phrase structure grammar where the feature values of the daughter nodes must unify to form a mother category, just as in the morphology rule above (3.1.1). In other words, the feature values put constraints on what constituents are legal. This is of course easily implemented using Prolog unification. A (quite simplified) rule, corresponding to **NP --> Det NBAR**, might look like this:

```
syn(np_det_nbar,
    [np:[determined=y, name=n, reflexive=n|Shared],
    det:[common=G, sing=N, def=D],
    nbar:[subcat=[]|Shared]
    ])
        :- Shared=[def=D, common=G, sing=N].
```

The first argument of the predicate above is the name of the rule (**np_det_nbar**). The second argument is a list of three elements: The first is the mother category (**np:FeatureList**); the two following elements are its daughters (**det:FeatureList** and **nbar:FeatureList**), respectively. The two daughter nodes must unify (agree) with respect to number (**sing=N**), gender (**common=G**) and definiteness (**def=D**). (These features are instantiated with the values given in the lexicon and passed on by the morphological analysis.)

The philosophy behind the grammar is to make it as simple as possible and put as much information as possible in the lexicon. An important concept is that of *subcategorization*, or what obligatory adjuncts words expect. By incorporating information concerning subcategorization in the lexicon, there have to be no explicit rules dealing with this in the grammar. A very small set of rules can be applied to all verbs, given the **subcat** info of each verb entry. For example, **subcat=[]** means that this entry cannot subcategorize for anything. If the entry happens to be a verb, what this means, is that the verb is intransitive. If the **subcat** feature of a verb entry is not the empty list, the parser will try to match a constituent with the value of the **subcat** feature and form a verb phrase consisting of the verb and matching complement(s). For an ordinary transitive verb, subcategorizing for a noun phrase (**np**), the feature setting would be **subcat=np** (it does not look quite this simple in the real lexicon, however). Furthermore, the lexicon must contain every possible **subcat** feature for every verb, a fact which results in a plenitude of entries for each verb. Here are two (simplified) examples of possible lexical items, the intransitive *resa* ('travel') and the transitive *äta* ('eat'):

```
lex(res, [v:[vform=impera, synmorphv=2, subcat=[]]]).
lex(ät, [v:[vform=impera, synmorphv=irreg, subcat=np]]).
```

The value of **vform** is the tense of the verb, **synmorphv** tells what conjugation a verb belongs to.

3.1.3.1 Gaps

To manage question structures and topicalized sentences, the grammar allows for certain movements to take place, which leaves traces, or *gaps*. When the parser tries all suitable rules, and finds a constituent containing a gap, it tries to find the moved constituent and unify it with the gap. The feature **gaps** is used to pass on information about moved constituents, to ensure that empty productions will only be produced when there really has been a movement [Gambäck *et al* 1994, p. 37]. For these rules to work, it is necessary that the feature **gaps** has the value **g(A,A,B,B)**, variables, that is. The default value is otherwise **g([],[],[],[])**. (For details on feature values, see 4.4.)

3.1.4 CLE semantics

The ultimate goal of the CLE is to give a semantic representation of the input language, and to this end, the people behind the Engine have developed the QLF, or Quasi Logical Form, which is an extended first-order predicate logic, more or less. The QLF is called a *quasi* logical form since it is not a fully fledged logical formalism, in the sense that it leaves, for example, quantifiers unscoped and anaphoric terms (pronouns, etc) unresolved.

The semantic module of the CLE can be said to be parallel to the syntactic one, in that there for each grammar rule and for each lexical item, is at least one specific semantic rule and at least one sense entry, respectively. To establish the "meaning", or at least to construct a QLF, for an input sentence, the sense entries of each word are put together compositionally by applying the semantic rules corresponding to the syntactic ones involved in the syntactic analysis of the same sentence. For each sentence, all possible QLFs are produced. Later processing steps can turn QLFs to logical forms proper, if there is a need for this, as it is, for example, in a database query system [Gambäck & Ljung 1993]. The QLF format seems to be quite adequate for machine translation tasks as it is, however [Alshawi *et al* 1991]. There are still other modules for checking the plausibility of competing QLFs, so that the most promising semantic interpretation can be selected.

3.2 Text source

To be able to use the CLE for parsing large texts without actually having to build a huge lexicon, we use a corpus of Swedish in which all words are provided with explicit grammatical information.

3.2.1 The Stockholm-Umeå corpus

We get the text sources for using as input to the CLE from the SUC core corpus [Källgren 1990, Källgren 1991, Källgren 1993]. SUC is a cooperation project between the departments of linguistics at the universities of Stockholm and Umeå, and the aim is to develop methods for large-scale grammatical analysis of unrestricted text. The project will eventually result in two text corpora for Swedish.

The project begun in 1989, and to be able to accomplish it in a five year period, it was necessary to recycle as much material and tools as possible. Not needing to build a lexicon was one of the capital issues, and the Swedish version of the TWOL-lexicon[1] of the University of Helsinki filled this need. To save time, all text material would have to be in a computer readable format from the beginning, and this fact has influenced the selection of texts.

To make the corpora as useful and readily available as possible for further research, much work has been put into the copyright questions.

3.2.2 The core corpus

The core corpus is a balanced one - which means that the texts cover many different styles of Swedish - and contains about one million words divided in 500 blocks of 2000 words each. The texts are selected from different genres following principles used for the Brown and the LOB corpora for English, which both have been a source of inspiration and knowledge for the Swedish corpus

---

[1] This lexicon is presented in: Koskenniemi, K. (1983). "Two-level Morphology: A General Computational Model for Word-form Recognition and Production", University of Helsinki, Publications of Dept. of General Linguistics No. 11.

builders. The blocks in the SUC corpus do not necessarily contain exactly 2000 words, but stop at the first natural cut after word 2000, thus forming a coherent subpart of the source (The Brown and LOB corpora cut the text at the first sentence boundary after word 2000). The SUC people can guarantee almost 100% tagging correctness, based on both automatic disambiguation and manual proof-reading.

3.2.3 The reference corpus

The other corpus, called the reference corpus, whose magnitude is not yet decided, but may end up as big as 10 million words or more, will contain the rests of the texts after the excerption of the core corpus blocks. For this additional corpus there is no guarantee of correctness, as it will only be automatically tagged.

3.2.4 The structure of SUC

The genres included in the core corpus are:
(the figures in parentheses indicate how many blocks of 2000 words each genre has)

Press: reportage (44)
Press: editorial (17)
Press: reviews (27)
Skills, trades and hobbies (58)
Popular lore (48)
Biographies, essays (26)
Miscellaneous (70)
Learned texts (84)
Fiction (126)

The LOB and Brown corpora have a special subcategory for religious texts. These kinds of texts are present in the SUC as well, but are found under different headings: "Skills, trades and hobbies", "Popular lore" and "Learned texts" [Källgren 1991].

3.2.5 The tags of SUC

The words in SUC are morphologically tagged in a theory independent way, to make the corpus as useful as possible for different areas of linguistic research. All word occurrences are followed by their lemma (i.e., the uninflected form), a two-letter long part of speech tag and none or several three-letter morphological tags depending on the categories. The morphological tags are unambiguous; a feature always gets the same tag regardless of what category it belongs to. For example, definiteness is always tagged "DEF", and its opposite paradigm value is tagged "IND". A noun entry looks as follows:

```
("<cykelpumpar>"
    ("cykel_pump" NN UTR PLU IND NOM))
```

NN means that the word is a noun; UTR says that it belongs to the common gender; PLU that it is the plural form; IND indicates that it is the indefinite form and NOM that it is in the nominative case (not genitive, that is). The underscore in the lemma indicates that *cykelpumpar* is a conjoined word formed by the two words *cykel* ('bicycle') and *pump* ('pump') - this kind of compound word forms is very common in Swedish.

The SUC tag set of the text we have been working with contains 21 different category tags of two letters and 24 morphological tags of three letters. In addition, there are three supplementary category tags for foreign words, abbreviations and numbers. The tags are always in the same order, depending on the category.[2] The tags will be presented in greater detail below.[3]

---

[2] For further information of the SUC tagging system see [Ejerhed *et al* 1992].

[3] In the final version of SUC, yet to be published, there will be a few minor changes to the tags we have been working with so far.

4 The present work

What our project has resulted in (so far), is a programme that makes it possible to parse SUC texts in the CLE. The greatest problem is how to deal with the fact that information expected by the CLE is not always possible to retrieve from a SUC text.

The idea has been to develop our programme - and at the same time to test the CLE grammar - with the help of a small text sample and then test it on unseen text. We have worked with a text of 130 sentences of variable length. While writing our programme, we have worked interactively with Björn Gambäck, who has improved the Swedish CLE grammar to increase its coverage.

4.1 Overview of the programme

The programme is written in SICStus Prolog [Andersson *et al* 1993]. When calling the start predicate (**start_suc_to_cle/6**) the user has to specify the following:

- The **time_out** time in seconds.
- The name of the input file (a SUC file in Prolog format).
- The name of a file containing all the infinitive and imperative verb forms for all the verbs in the CLE lexicon (See 4.3.2).
- The basis of the file names in which the output lexica and analyses will be written.
- An extension in digits (e.g. a date) for the output files.
- The lexicon strategy.

The programme takes as input sentences from a SUC text and for each word in a sentence, CLE lexicon entries based on the information given in the SUC text are created. Two new lexica are created for each sentence, one containing only the words missing in the CLE and one containing all the words in the sentence regardless of if they are found in the CLE lexicon or not. When entries for all words in a sentence have been provided, the new additional lexicon is loaded into the CLE, which produces a syntactic analysis of the sentence. How many parse attempts and what lexica will be used is defined by the lexicon strategy chosen. The procedure is repeated

for every sentence in a text. All syntactic constituents produced by the parser are saved in a file. This file is subsequently going to be used for building (disambiguated) syntactic parse trees.

While the entire CLE system comprises several different modules, as explained above, we are only concerned with the syntactic parser of the Swedish CLE.

4.2 Parsing SUC texts in the CLE

To be able to use SUC texts as input to the CLE, we must transform the SUC tag sets to equivalent CLE lexicon features, and since the LISP-like format of the SUC corpus does not suit a Prolog implementation, first of all the SUC text has to be transformed into Prolog terms, which is done by a separate programme. This programme takes one word at a time from the SUC text and makes a Prolog fact of it, whose functor is either **ord/2** if a word, or **tecken/1** if an orthographic sign.
Input:

```
("<{mnesbyte>"
      ("{mnes_byte" NN NEU SIN IND NOM))
("<$?>")
```

Output:

```
ord('{mnesbyte',['{mnesbyte',[nn,neu,sin,ind,nom]]).
tecken('$?').
```

4.2.1 Creating sentence specific lexica

To be able to analyze a sentence, the CLE must have access to a lexicon in which all the words in the sentence are found. The original CLE lexicon contains morphological, syntactical and semantic information. For our purpose, only syntactical and morphological information is of interest.

The Swedish lexicon that the CLE has access to is too limited and too domain specific to cover unrestricted text. The function words (e. g. prepositions, determiners, pronouns) and auxiliary verbs, are all - or most of them - found in the CLE lexicon, but most

members of the open categories (nouns, adjectives, verbs, etc) are not (not only due to the fact that these categories are in principle infinite; the lexicon contains no more than some 1 500 entries). For this reason, we decided to create new, sentence specific, lexica for every sentence in the SUC text.

This is done by transforming the tag set each word in the corpus is furnished with into equivalent CLE features. The example word above (4.2) might end up as something like this after being transformed into CLE format:

```
lex('{mnesbyte',
      [nbar:
        [common=n, @takes_infix(n), simple=y, @gap_island,
         subcat=[], def=(n\/poss), @relagr(K), agr(K), sing=y,
         person=3, conjoined=n, genitive=n, synmorphn=irreg,
         measure=_, lexform='{mnesbyte']]).
```

(For further details, see 4.4.)

These new entries do not contain any semantic information, so the new lexicon is not of general use - it works for syntactic parsing only. Another reason as to why the new lexicon cannot be of general use is that the new entries are based on the actual occurrences of the words, not the uninflected, "neutral" or primary forms.

4.2.2 The lexicon strategy

As said before, *two* lexica are produced for each sentence, one which only contains words not found in the original CLE lexicon (called *first* lexicon), and one in which all words in the sentence get at least one fresh entry (called *second* lexicon).

For every word in a SUC sentence, new CLE entries are created. Each new entry is looked up in the original CLE lexicon, and if found, the new entry is *not* put into the first lexicon, only in the second. The look up is performed by the CLE system predicate **external_lrecord/3** which succeeds if a word is found in the lexicon, and returns the lexicon entry/entries for that particular word. With the help of this predicate we can compare newly

created entries and those in the original lexicon to see if they match. The fact that the CLE entries will always be more complete than those made by our programme is a good reason for using the CLE lexicon as far as possible, though, as we will see below, there are cases when the original entries are not general enough.
There are four different lexicon strategies to choose from. When starting the user can specify what lexicon(s) will be used. When the lexicon strategy is *first_lexicon* , the programme will only attempt a parse with help of the first lexicon. If the strategy is *second_lexicon*, only the second lexicon will be used. For both these strategies the program will attempt to parse the sentence once. When the strategy is specified to *both*, two parse attempts will take place, one with help of the first lexicon and one with the second. The forth strategy, the default strategy, is called *both_if_needed*, here if the system fails to parse the sentence with the help of the first lexicon, the second lexicon will be loaded and the system will make a new attempt.

This double lexicon strategy is due to the restrictions some entries have in the CLE lexicon. For example, the entry for *arbete* ('work') has a feature **mass=n**, which means that it is a count noun. A phrase like *söka arbete* ('look for a job') will be impossible to parse because of the fact that the CLE grammar rules demand that a count noun is of the definite form or have an article. In the second lexicon, where all words get entries based on SUC tags, *arbete* will occur with the same feature uninstantiated, since the default value for nouns is **mass=_** (a noun might or might not be countable, in other words).

The double lexicon strategy results in some new successful parses that would not have been found with just one lexicon, but at the same time, the processing time for a text increases quite a lot. In general, the time it takes to create and load new lexica is negligible compared to the time the syntactic analyses take.

4.2.3 Sometimes it fails

Since the new lexica do not always contain the information the CLE expects (and since the grammar is still being worked on) there seem to be sentences for which the parsing does not terminate. For this reason, there is a "time-out" after an optional amount of time, so that the next sentence can be processed instead.

4.3 Transforming SUC tags to CLE features

A new entry, produced by our programme, contains all information given in the SUC tag set, plus some additional features. For example, all nouns (**nbar**) are given the default value **subcat=[]**. This means that nouns cannot subcategorize, i. e., they are not supposed to take obligatory adjuncts. The reason as to why we have to set certain feature value pairs apart from those directly derived from the SUC tag set, is that the CLE sets its own default values for each feature value pair missing in a lexicon entry. The default values given by the CLE are more often than not the anonymous variable, a fact which will result in an unhappy number of syntactic analyses, since the anonymous variable means that there are no restrictions on what values a feature can take. We tried to parse a sentence of five or six words, all missing in the original CLE lexicon, with the help of a new lexicon containing only features directly corresponding to the tags given in the corpus. The new lexicon contained only one entry for each word in the sentence. The result was disturbing, to say the least: About 13 500 analyses were found!

By carefully choosing our own default values, the number of analyses is drastically reduced to a quite manageable level. The risk is that words that fall outside the standard pattern are not given correct entries. If a noun that actually subcategorizes for something turns up in a sentence, there is no way for us to deal with this. The only solution is to carefully examine the CLE´s analyses, and when "strange" words turn up, add the correct entries to the CLE lexicon and redo the parsing (there already exist a lexical acquisition device for the CLE [Gambäck 1992]). Hopefully, the default values we have chosen are correct in most cases (this remains to be seen, however).

4.3.1 The verb exception

The method above works for most words except verbs, and there are two reasons for this: 1) The CLE lexicon and SUC consider different verb forms to be the primary form: In SUC the infinitive is considered the primary form, but in the CLE the imperative; 2) The CLE grammar expects information about the subcategorization of the verbs from the lexicon, but the SUC tagging, on which the new entries are based, does not have any such information to offer. To create proper verb entries, the sentence in which the verb occurs has to be searched for certain specific words indicating that the verb might be of some specific verb type.

To solve the first of these two problems we have made a special "infinitive/imperative lexicon". What this means is that for every verb in the CLE lexicon, there is a Prolog fact (**verb/2**) with the infinitive form of the verb as the first argument and the imperative as the second. This "lexicon" is consulted when the programme starts, and makes it possible to look up a verb in the infinitive by checking if there is an imperative counterpart to be found in the CLE lexicon. (The facts look thus: **verb(komma,kom)**, **verb(regna,regna)** etc, and are created by a separate programme.)

The handling of the second problem is dealt with in greater detail below (4.4.3), but this is the gist of it: When the programme runs into a verb, all words following that verb are checked to see whether there are any words indicating that the verb might be of some special verb type (particle or reflexive verb, etc). If this is the case, new entries are created, and each of these are looked up in the original lexicon (with the help of the "infinitive/imperative lexicon"). All possible verb variants will be created - if for example what might be a reflexive particle verb is found, a reflexive particle verb entry, an ordinary particle verb entry and a reflexive verb entry will be created. If no entries with matching **subcat** features are found in the CLE lexicon, the new entries are added to both lexicon files. When all words in a sentence have been looked up, the new lexicon file is loaded into the CLE and the sentence is parsed, as explained above. All lexica are saved and can be inspected after running the programme.

4.4 The SUC tags and their CLE values

As already mentioned, the SUC tag information is not enough for completing the CLE features. The problem of handling the cases when SUC tag sets and CLE features do not match has been solved in different ways depending on part of speech category.

4.4.1 Morphology tags

A SUC tag set contains a two-letter part of speech tag followed by a set of three-letter morphological tags. There is almost a one-to-one relationship between the morphological SUC tags and their CLE counterparts.

These are the tags and their CLE values:

UTR (Gender)	**common=y**
NEU (Gender)	**common=n**
MAS (Gender)	**common=y**[4]
SIN (Number)	**sing=y**
PLU (Number)	**sing=n**
DEF (Definiteness)	**def=(y\/poss)**[5]
IND (Definiteness)	**def=(n\/poss)**
NOM (Case)	**genitive=n**
GEN (Case)	**genitive=y**
SMS (Compound)	Has no counterpart.[6]

---

[4] Most human nouns in Swedish belong to the common gender, the masculine form (-e ending for adjectives) is an optional form used for male nouns but this form cannot be combined with non common nouns. Thus, *the little boy* may be translated 'den lilla pojken' or 'den lille pojken', both are grammatically correct but *the little child* may only be translated to 'det lilla barnet'. The CLE system currently has no feature for this.

[5] The **poss** value of the feature **def** allows the unification process in possessive NPs. E.g. in a phrase as *min vackra bil* ('my beatiful car'), where *bil* is indefinite but *vackra* is definite.

[6] This tag is used for compound nouns sharing the second component (the head noun) with another noun, as in *kvinno- och mansgrupper* ('women's and men's groups'). Here *grupper* ('groups') is the shared second element of

PRS	(Verb form)	**vform=(present/\fin)**
		vform=(presp) (After PC (participle))
PTR	(Verb form)	**vform=(imperf/\fin)**
INF	(Verb form)	**vform=(inf)**
SUP	(Verb form)	**vform=(supine)**
IMP	(Verb form)	**vform=(impera)**
KON	(Mood)	Has no counterpart, this information is lost.
PRF	(perfect form)	**vform=perfp**
AKT	(Voice)	**passive=n**
SFO	(Voice)	**passive=y**[7]
POS	(Degree)	**aform=norm**
KOM	(Degree)	**aform=comp**
SUV	(Degree)	**aform=sup**
SUB	(Pronoun form)	**case=subj**
OBJ	(Pronoun form)	**case=nonsubj**

In the cases where there is more information in the SUC tag set than needed in the CLE, this information is added anyway; the system ignores strange features as it adds the default values for the features missing.

Words that do not inflect and mostly occur in fixed expressions get no morphological tags, they are tagged with part of speech tag only. The morphological tags are substituted for dashes. As, for example, the word *synes* in the expression *till synes* ('apparently'):

```
("<synes>"
        ("synes" NN - - - - ))
```

In these cases, the new CLE entry will get default values set for the category.

---

the two compound nouns. In the present version of the programme this tag will disappear, but new grammatical rules are needed to deal with this very important information.

[7] This tag is used for all verb forms ending with -s, whether it is a passive use of the verb or not. Most cases will be passive forms anyway, so we decided

4.4.2 Macro calls and defaults

As said before, the system sets its own defaults for the feature value pairs not declared, so the lexicon entries will always be complete, in the sense that no feature value pair is lacking. The default values are often the anonymous variable, which means that the feature may take all possible values and that the system will try them all while trying to parse the constituent. To avoid this we have declared our own defaults for the information missing in the SUC tag set. Many of our choices are made on intuitive and quantity basis, that is, we have chosen the values we believe will be right for most of the words. These values will be presented below under each category section as they vary depending on the category.

As can be observed in the example above (4.2.1), in a lexicon entry there may occur instances headed by an @, and these are called 'macro calls' and are used for abbreviating the settings of specific feature values. A macro call is expanded by the system to a (or a set of) specific feature value pair(s). With the help of the macro calls one avoids writing all features and their values in the lexicon and the grammar rules. The macro calls can instantiate the default values for a subcategory that separates its members from the members of the mother category or other subcategories. Thus proper nouns (names) is a subcategory of NP, separated from all other NPs by the feature value pair **name=y**. For example, the macro call **@proper_name(Genitive)** will be expanded as follows:

```
@proper_name(Genitive) >
    [np:
      [type=norm, simple=y, name=y, gaps=g(A,A,B,B),
       passive=_, reflexive=n, common=y, sing=y, person=3,
       takes_infix=n, def=n, conjoined=n, genitive=Genitive]]
```
[8]

---

to transfer it into the **passive=y** feature value pair. The -s ending is also used in reciprocal meaning and for deponent verbs.

[8] The feature **reflexive** has the value **y** (yes) when the NP is a reflexive pronoun. **passive** indicates whether the NP may function as the subject of a passive sentence or not. All features will be presented in section 4.4.3.

This means that in the lexicon we only need to write:
lex(stockholm, @proper_name(n)).

4.4.3 Part of speech tags

Open categories: Content words

NN - Nouns

Tagged NN in the SUC corpus, the nouns are denominated **nbar** by the CLE. SUC provides information about gender (the neuter or the non-neuter), number (the singular or the plural), definiteness (definite or indefinite) and case (the genitive or the nominative). There are many more CLE features for nouns, most of them concerning morphological and syntactical subcategorization or CLE-specific features needed for the system to work. The system itself sets the default values when features are not specified, so we started out by specifying the default values that did not agree with the ones set by the system. The features and macro calls we add for nouns are the following:

simple=y
subcat=[]
person=3
conjoined=n
measure=_
lexform=Lemma
synmorphn=irreg
@takes_infix(n) > **takes_infix=n**
@gap_island > **gaps=g(A,A,B,B)**
@agr([C,S,P]) > **[common=C, sing=S, person=P]**
@relagr([C,S,P]) > **[relcommon=C, relsing=S, relperson=P]**

The **simple** features indicates whether we are dealing with a non compound item or not. We give all nouns, whether compounds or not, the value **simple=y**, to make the system treat them as units and avoid attempts to segment words in possible compound items. When **conjoined** has the value **n** ('no') it means that this constituent contains no conjunction. As we are dealing with one-

word constituents **conjoined** will always be negative. **takes_infix** indicates if the word takes a special infix when participating in a compound form. **measure** has the value **y** for words as 'inch', 'meter', etc. **person** is instantiated as **3**, i.e. third grammatical person. The **relagr** macro call passes the agreement information from a relative clause. **synmorphn** indicates what declension the noun belongs to; we classify all as irregular (which means that the CLE morphology rules cannot be applied) because this information is missing in SUC. **lexform** is the basic, uninflected, word form and gets its value from the SUC lemma. The **@gap_island** macro call expands the gaps into variables (the default is four empty lists).

Given a SUC tag set, a lexicon entry with the above settings would look like this:

```
lex('{mnesbyte',
    [nbar:
     [common=n, @takes_infix(n), simple=y, @gap_island,
      subcat=[], def=(n\/poss), @relagr(K), @agr(K), sing=y,
      person=3, conjoined=n, genitive=n, synmorphn=irreg,
      measure=_, lexform='{mnesbyte']]).
```

PN - Names

In SUC, names are tagged PN and get only one morphological tag, the case information. In CLE they are classified as NPs, and there is a special feature for them, namely **name=y**. The case information is transferred to the **genitive=(y\/n)** feature. While working on our programme, Björn Gambäck introduced a special macro call for names:

@proper_name(Genitive)

This macro is expanded into an NP as shown before (4.4.2).

VB - Verbs

The verbs are divided into some fifty categories, according to transitivity, reflexivity, whether they are particle verbs or take some special preposition and so on.

Most of these fifty verb categories do not cause us much trouble, since many of them are semantical subcategories rather than grammatical ones. As already said, we use only the syntactic

module of the CLE, and do not care about semantic analysis (we leave the discussion about whether syntax and semantics can or should be separated for some other occasion). Nevertheless, there remain a few verb categories of syntactic importance. This is a problem, since some of the information needed to create new verb entries is impossible to retrieve from the tag set of the corpus alone.

Among other things, the CLE needs to know whether a verb is transitive or intransitive. If a verb can be both transitive and intransitive, there are two separate entries in the lexicon. There is no information about transitivity in the corpus. To solve this problem, we simply decided that each verb can be both transitive and intransitive, and accordingly, for each verb in the corpus, two entries, one transitive and one intransitive, are made by default. (We have not yet decided how to handle ditransitive verbs. The CLE of course expects ditransitive entries for ditransitive verbs.)

There are two other verb types that can be retrieved indirectly from the corpus: particle verbs and reflexive verbs. Particles belonging to a verb are marked **(AB)** in the corpus. **(AB)** actually means "adverb", but if a word is marked **(AB)** *and* is listed as a (core) preposition in the CLE lexicon, it is a verb particle and probably belongs to the preceding verb. In the sentence *titta efter om någon på gatan tog skada* ('take a look and see if anyone in the street got hurt') the particle verb *titta efter* ('take a look') will be found, because of the fact that *efter* will be tagged **(AB)** and furthermore found in the CLE lexicon as a preposition. If a particle is found, it is linked to the preceding verb, and a particle verb entry is created. (In the final version of the SUC, verb particles will be retagged and get a distinct particle tag, **PL** [Gunnel Källgren, PC].)

Reflexive verbs are searched for in a similar manner. Reflexive personal pronouns are not tagged in any specific way in the corpus, so if a pronoun is found that can be a reflexive pronoun (*mig* ['me'], *dig* ['you' - singular], *sig*[9] ['themselves/herself/himself/itself'], *oss* ['us'], *er* ['you' - plural]), a reflexive verb entry is created for the preceding verb. Thus the sentences *han tvättar sig* ('he is washing himself') and *han tvättar mig* ('he is washing me') will both give rise to reflexive verb entries.

---

[9] *sig* is always reflexive, actually.

The grammar will not over-generate reflexive analyses, which one might expect, however, because in the second case, where the verb is not reflexive, the grammar will rule out the reflexive interpretation, since the CLE grammar demands that there is agreement between the subject and the object pronoun.

The same method is used to create verbs of the categories particle reflexive verbs (*Hälla i sig en sup*) and reflexive particle verbs (*Slå sig på lingvistik*)[10]. As already mentioned, for each verb, a transitive and an intransitive entry are made by default.

This might seem like a somewhat crude way to handle it, but we are dealing with a veritable Catch 22: To be able to parse a sentence, we have to create a proper lexicon. To be able to create a proper lexicon, we would have to parse the sentence.

These features (and macros) are added for verb entries:
synmorphv=irreg
nullmorphv=n
gap=n

All "SUC verbs" are classed irregular (**irreg**). This means that they cannot be inflected, which just is a convenient way of by-passing the CLE morphology processing. No morphological analysis is needed, since all relevant information is found in the SUC tags. **nullmorphv=n** is a way of telling the CLE that the different tenses of a verb are not expressed with the same inflection, that the verb is unambiguous with respect to the tense, in other words. **gap=n** means that a word is not an empty production, a gap.

Intransitives (V_SUBJ):
@gap_island
subcat=[]

Transitives (V_SUBJ_OBJ):
subcat=[@np_object(Gaps,PassiveForm,n)]
Where **Gaps** unify with the **gaps=Gaps** feature of the verb. This feature is used for keeping track of empty productions.

---

[10] Examples due to Gunnel Källgren.

Reflexive verbs (V_SUBJ_REFL):
@agr(Agr)
subcat=[@np_reflexive(Agr)]
Agr makes sure that there is agreement between subject and reflexive pronoun.

Particle verbs (V_SUBJ_PARTICLE):
subcat=[@p_particle(Particle)]

The different **subcat** features can be combined to create new verb categories, for example **V_SUBJ_REFL_PARTICLE**, etc.

JJ - Adjectives

There are two principally different ways to compare the Swedish adjective. Either by inflection or by adding *mer* ('more') or *mest* ('most') in front of the positive form. In the corpus, positive adjectives (JJ) are tagged for gender, number, definiteness and case, while comparatives only have case information, and superlatives definiteness and case. Adjectives belonging to the third declension, that is 'more and most adjectives', are always tagged as positives in the text. The definite singular form and the plural forms (both definite and indefinite) coincide - they all have the ending -a. We have chosen to classify all adjectives as belonging to the third declension, i.e. "more/most" adjectives (**synmorpha=3**), to prevent the system from testing all declension and to be able to deal with comparative and superlative forms constructed with 'more' and 'most'. The defaults and macro calls added to adjectives are:
subcat=[],
@gap_island
synmorpha=3
conjoined=n
nullmorpha=n

The **nullmorpha** feature indicates what form, if any, has a null ending inflection. We have set the value to **n** because we are not interested in the system trying a parse where the adjective would have another form apart from the one given by the SUC tags POS/KOM/SUP. For example, the adjective *bra* ('good') has the

feature value pair **nullmorpha=all**, because all its forms are identical. An adjective entry will thus look as follows:

```
lex(smidig,
      [adjp:
       [aform=norm, common=y, sing=y, def=n, subcat=[],
        @gap_island, synmorpha=3, conjoined=n,
        nullmorpha=n]]).
```

AB - Adverbs

Adverbs are tagged AB in SUC. Adverbs derived from adjectives furthermore have comparative form information; adjectives and adverbs share the feature **aform** that gets its value from tags POS, KOM and SUV. The features added by the programme as default values are:
sentential=_,
@gap_island

The **sentential** feature says whether the constituent can modify a sentence or not. An adverb entry will look like this:
lex(smidigt, [advp:[sentential=_, @gap_island, aform=norm]]).
or like this:
lex('n{stan', [advp:[sentential=_, @gap_island]]).

RG - Cardinal Numbers

Cardinals are tagged RG, but there are two different tag sets, one for the singular forms *en* and *ett* ('one') and another one for plurals. The case information is lost when transferred to CLE format. For plurals it is done by the macro call:

```
@plural_number(_NumType)  >
      [number:
       [lexical=y, spelledout=y, sing=n,
        numtype=_NumType]]
```

The underscored variable **NumType** indicates the missing information about the magnitude of the number (for example digit, hundred). The feature **spelledout** indicates whether the number is

written with letters (*fem*, 'five') or not (*5*) and **lexical** says if the entry is a lexical item or not.
For the singular cardinals (*en* and *ett*) there are two special tag sets marking gender, because Swedish demands concord in gender between 'one' and the head noun. This information is transferred to a CLE format by the macro call:

```
@singular_number(Gender, digit) >
                [number:
                 [lexical=y, spelledout=y, sing=y,
                  common=Gender, numtype=digit]]
```

Not spelled out numbers are tagged NR in SUC, and are transferred into CLE by the macro call:

```
@proper_number(_NumType) >
      [number:
       [lexical=y, spelledout=n, sing=n,
        numtype=_NumType]]
```

RO - Ordinals

The ordinals (RO) have only one morphological tag, the case tag. This information is lost when transferred into CLE format. A special macro call with no feature transport does it:

```
@ordinal_number >
          [degree:
            [aform=sup, adjform=sup, @gap_island, upmod=n]]
```

Ordinals are, as can be observed, classified as superlative degree adverbs that can not modify VPs (**upmod=n**). The **adjform** feature inherits its value from the adjective **aform** in "more/most" adjective constructions. While the **aform** will indicate the form of the whole adjective phrase, the **adjform** will still be the actual form of the adjective. As ordinals have both features instantiated to **sup**, they can only be combined with superlative adjectives (*tredje största stad*) in this kind of phrase.

There are no entries in the CLE lexicon for the adjectival use of ordinals (*tredje staden*). There is a rule in the grammar that allows for an adjectival use of this kind of **degree** elements, however.

Ordinals also have a special tag set for the singular and dual tokens (which is not mentioned in the presentation of the annotation system of SUC), but all that information is lost as all ordinals have the same macro call in the CLE.

IN - Interjections

Being tagged with a single part of speech tag (IN), interjections do not cause any serious trouble when it comes to transferring into CLE format. The macro call **@interjection** takes care of these, and it will be expanded into:

```
[interjection:[]]
```

Closed categories: Function words

The categorization of function words is not always clear-cut. Many of these words can have different uses, whether allographs or not. How to classify a word also depends on what the grammatical rules look like. These words should, in principle, be found in the CLE lexicon, but to be able to compare different categorizations we have chosen to have them transferred from the SUC material anyway. Another reason is that if some function words turn out to be missing, this way it is possible to parse the sentence anyway.

HA - Interrogative/relative adverbs

(*när, var, vart, hur, då, där, som*)

Interrogative and relative adverbs are tagged HA in SUC, as a special category, and are transferred to CLE features by the special macro call:

```
@positional_adverb >
    @pronominal_adverb((q\/r), _Wh) >
            [pp:[@gap_island, type=(q\/r), wh=_Wh]]
```

The first macro call was designed specially for transferring SUC entries into CLE ones. The second macro call is used for transporting the values of the pronominal adverbs, such as demonstratives (*här*,

'here'), interrogatives (*var*, 'where'), relatives (*då*, 'then') and indefinites (*alltid*, 'always'), that will end up as PP[11].
The first argument is the value of the feature **type**. As SUC does not differentiate relative from interrogative adverbs, **type** has to be instantiated as a disjunction; **q** for interrogatives or **r** for relatives. This was necessary because the **type** information is essential for CLE, but it is missing in the SUC. The sentence inherits the **type** information. When the **wh** feature has the value **y**, it indicates that we are dealing with an interrogative element.

PN - Pronouns

SUC does not make any difference between the personal and indefinite pronouns at category level, both are tagged PN. They are differentiated with the help of the definite/indefinite opposition - definite for personal pronouns. The CLE lexicon has two different macro calls, one expands the personal pronouns into NPs and the other the indefinites into determiners:

Personal pronouns

```
@personal_pronoun([Gender, Number, _P], _Ref, Case)  >
    @np_pronoun([Gender, Number, _P], _Ref, Case, norm) >
          [np:
           [common=Gender, sing=Number, person=_P,
            reflexive=_Ref, case=Case, nform=norm, pron=y,
            def=y, type=norm, name=n, simple=y,@gap_island,
            determined=n, mass=n, conjoined=n]]
```

The **nform** feature has two values: **det** for impersonal subjects and **norm** for all the others. **det** is used in cases as *det regnar* ('it's raining'), when the subject only fills a grammatical need.

The **determined** feature has the value **y** (yes) if there is a determiner inside the NP, which of course is not the case for the personal pronouns. The **pron** feature separates the NPs that are

---

[11] In the CLE grammar most mobile constituents are classified as prepositional phrases and dealt with by a small set of rules.

pronouns from the ones that are not, for example names and "common" NPs built by a noun and optional dependents.

Indefinite pronouns

```
@det_indef_pronoun([Gender, Number, _P], _Mass) >
    @determiner([Gender, Number, _P], _Mass, n) >
        [det:
            [common=Gender, sing=Number, person=_P,
             mass=_Mass, def=n, nbarell=y, type=norm,
             lexical=y, @gap_island, simple=y, conjoined=n]]
```

The **nbarell** feature indicates if a determiner alone can appear as an NP (with an elliptical NBAR). As can be observed, both macro calls transport the gender and number information and for personal pronouns even the case information. In the SUC tag set there is no information about reflexivity, person or mass, so these values remain uninstantiated.

There is not a one-to-one relation between SUC and CLE concerning the indefinite pronouns, but as most function words they should be in the CLE lexicon. For the *first lexicon* new entries will only be created for the words missing, for example special (dialectic) variants and similar exceptions.

HP - Question and relative pronouns
(vem, vilken, vad, vilket, vilka)

In SUC, question and relative pronouns are tagged as one category, HP. In CLE they are classed NP with different values of the **type** feature, **q** for questions and **r** for relatives.

```
@interrogative_pronoun([Gender, Number, 3], (q\/r)) >
    [np:
        [@agr([Gender, Number, 3]), type=(q\/r), sentential=_,
         passive=_, nform=norm, pron=y, def=y, name=n,
         simple=y, reflexive=n, @gap_island, determined=n,
         mass=n, conjoined=n]]
```

The feature **passive** indicates whether the NP can be the subject of a passive sentence or not.

The Swedish word *som* is ambiguous between subordinating conjunction (SN) and relative pronoun (HP) and consequently it is tagged differently depending on the specific context. When it has HP as its unique tag, it corresponds to an NP entry with a special set of features.

PS - Possessive pronouns

Possessive pronouns (tagged PS in SUC) are treated as determiners, and have a special macro call:

```
@possessive_pronoun([Gender, Number, _P]) >
        [det:
          [common=Gender, sing=Number, person=_P,
           mass=n, def=poss, nbarell=y, type=norm,
           lexical=y, @gap_island, simple=y, conjoined=n]]
```

HS - Question and relative possessives
(*vems, vilkets, vilkens, vilkas, vars*)

Once again SUC does not make any difference between question and relative elements, the possessives always have the tag set HS DEF. CLE has a special macro call for these words:

```
@wh_determiner(Type, Agr, Mass, Lexical, Simple, Nbarell)
```

We call upon it with the following values:

```
@wh_determiner((q\/r), [_, _, 3], _, y, y, _) >
        [det:
          [common=_, sing=_, person=3, mass=_, def=n,
           nbarell=_, type=(q\/r), wh=y, lexical=y,
           @gap_island, simple=y, conjoined=n]]
```

Curiously, all question and relative possessives always have the feature value pair **def=n** (indefinite), while they can only be DEF (definite) in SUC.

DT - Determiners

The determiners do not cause any real problem, but as said before, the classification of function words is not always clear-cut, and therefore there are many differences between the two systems. Tagged DT in SUC, there is information about gender, number and definiteness that can be easily transferred to a CLE entry by the macro call:
@determiner([Gender, Number, _Person], _Mass, Def)

The expansion of this macro call is shown in the pronoun section.

HD - Question and relative determiners

(*vilken, vilket, vilka*)

As is the case with other categories, SUC does not make any difference between question and relative determiners: They belong to the same category tagged HD. HD words have gender and number information and, unlike the question and relative possessives, these determiners are always indefinite, a fact which coincides with the point of view of the CLE architects.
Here we once again use the macro call **@wh_determiner**, now with the following values instantiated:
@wh_determiner((q\/r),[Gender,Number,3], _, y, y, _)

In this case we have the gender and number information (c.f. HS above).

PP - Prepositions

These do not cause any problem. Tagged as PP in SUC, they have a macro call **@core_preposition** in the CLE. There are other categories classified as prepositions in the CLE, for example subjunctions, but the **@core_preposition** macro deals with "real" prepositions only. In SUC, prepositions that function as verb particles are also tagged AB (i.e. adverb), a fact which is utilized when making particle verb entries (see above). The expanded macro call will look as follows:
[p:[subordinator=n, conjoined=n]]

The **subordinator** feature separates the prepositions from the subordinating conjunctions, also classified as **p** in the CLE.

KN - Conjunctions

All coordinating conjunctions are tagged KN in SUC. The CLE function word lexicon has different entries depending on the constituents that are to be conjoined. There are two entries for *och* ('and'), six for *eller* ('or'), one for the comma, etc. The information needed is not available in the SUC tags, so we have made entries with the category and the lexical form only. Thus all conjunctions should have 'hand made' entries with all the information required because of their importance to sentence structure. A conjunction entry made by the programme will look like this:
lex(Conj, [conj:[lexform=Conj]]).

SN - Subordinating conjunctions

As mentioned before, the subordinating conjunctions (tagged SN in SUC) are in the CLE system grouped together with the prepositions, separated from these by the feature **subordinator=y**.

```
@subordinating_conjunction(_HasComp) >
        [p:
          [subordinator=y, conjoined=n,
           hascomp=_HasComp]]
```

If **hascomp** has the value **y**, the subjunction is to be followed by 'att'.

There is a problem recognizing subjunctions longer than one word. For example, *för att* is tagged **för/PP att/SN**, but it is the two-word cluster that functions as subordinating conjunction and not only the word *att*, actually tagged as SN. From the CLE point of view, *för* should be the conjunction and *att* its complement.

5 Results, conclusions and future work

5.1 Results

The main task of the present work was to parse SUC texts in the CLE. In order to do this, we have had to transform the information in the SUC tag set into a CLE format. Results of this can be seen in appendix A, where a sample lexicon file is found.

When testing the programme (in *first_lexicon* mode, see 4.2.2) on 130 input strings, most of which were proper sentences (a few were phrases rather than sentences, for example: *blickkontakt och turtagning*), 23 input strings got at least one complete analysis; 28 failed because of time-out (20 minutes); 68 strings were not given any complete analysis at all; 11 failed because of problems segmenting certain words. Segmentation problems occur sometimes when running the programme in *first_lexicon* mode. The reason is that the lemmas (or dictionary headword forms) of the words in the SUC text are used for the look up in the CLE lexicon. Sometimes this will fool our programme: If the lemma is found in the CLE lexicon, but the occurrence of the same word in the SUC text cannot be derived with the help of the CLE morphology, then the system will not recognize the word. For example, the lemma of *andre* is *andra*. The programme will find *andra* in the CLE lexicon, but the word *andre* of the input string will get no morphological analysis, because of the fact that the CLE currently has no rule for this. When running the programme in *second_lexicon* mode, this problem does not occur.

Of the 23 strings that got complete analyses, 17 of these got 6 or less different readings, three got 8-9 readings, and three strings got more than 70 readings (*De möter inte läkarens blick, de fäster istället blicken obestämt i fjärran*; *Läkaren å sin sida koncentrerar sig mycket bestämt på sin undersökning och undviker att se på patientens ansikte* and *Du kan också minska ögonkontakten med den du talar med eller slänga en blick på klockan*). For a few examples of how the output may look like, see appendix B.

When running the programme in *second_lexicon* mode on the same text, four more sentences were given complete analyses (with 4, 36, 72 and 768 readings respectively - the latter for the sentence *Med blicken uträttar vi alltså mycket*), 53 sentences were not given any complete analyses and 52 sentences failed because of time-out. The *second_lexicon* strategy can handle a few more sentences, but on the other hand, for each sentence a greater number of analyses is produced.

Only 27 complete syntactic analyses out of 130 sentences might sound like a rather meagre result, but the fact is that partial parses are possible to retrieve for most of the failed analyses. It should be possible for an annotator to put these partial parses (constituents) together, given the right tools for doing this.

5.2 Conclusions

With the help of the programme presented in this essay, it is possible to parse SUC text in the CLE, which of course is a prerequisite for using the CLE grammar for building a treebank based on SUC material. The programme can furthermore be of use for debugging the CLE grammar, since trying to parse unrestricted text might be an efficient way of finding things the grammar "ought" to be able to handle but does not.

To link the two systems of language description, that of SUC and that of the CLE grammar, was not as easy as it seemed at first glance. Both SUC and CLE use rather theory independent classification principles, although the classification of function words differ. To transfer morphological SUC tags into CLE features was not too problematic, the difficulty was to transfer the classification of, above all, function words and to add the category specific default values.

We have learnt that a SUC text is *not* a CLE lexicon of the same text, but can be turned into something that resembles one by transforming the SUC tags to equivalent CLE feature value pairs and by adding default features when the SUC text lacks vital information. Furthermore, some information not found explicitly in a SUC text can be extracted by searching for things that point in a certain direction. For example, if a particle is found, the preceding

verb is most likely a particle verb, and if a (potentially) reflexive pronoun is found, the preceding verb might be a reflexive verb.

The major difficulty has been to decide what default information should be given the new CLE entries in those cases where the SUC tags lack information expected by the CLE. The thing is to find a balance between not being too specific and not being too general. If too much information is put in the new lexicon, this might constrain the grammar, so that proper analyses will not be found. Lack of lexical information, on the other hand, will lead to an overgeneration of syntactic analyses (and time-out, in the worst cases).

To tackle the problem, we have adopted the strategy of repeated parsing attempts with decreasing amount of information in the lexicon. If a first parsing attempt fails, this might be due to the fact that entries in the original CLE lexicon are too specific for the sentence at hand, so a second parsing is tried, in which all words get other, more general, entries produced by our programme.

A problem this approach cannot handle, is when less prototypical words, which fall outside the classification of the default values given by our programme, turn up. How great this problem is, can only be appreciated after running the programme on a bigger sample of text material and thoroughly analyzing the result.

5.3 Future work

The next step is to build phrase structure trees of the output of the CLE´s syntactic parser. To this end, some kind of user interface has to be developed to help annotators choose the right analysis and/or correct faulty ones. The output of the CLE must probably undergo a few changes; some of the peculiarities of the CLE grammar might not be welcome in a treebank meant for general use. Furthermore, some clever mechanism for taking care of cases falling outside the scope of the CLE grammar has to be built. Parallel to this, the programme presented in this essay and the CLE grammar have to be debugged and refined.

References

[Agnäs *et al* 1994]
Agnäs, M-S., H. Alshawi, I. Bretan, D. Carter, K. Ceder, M. Collins, R. Crouch, V. Digalakis, B. Ekholm, B. Gambäck, J. Kaja, J. Karlgren, B. Lyberg, P. Prince, S. Pulman, M. Rayner, C. Samuelsson, T. Svensson (1994). "Spoken Language Translator: First Year Report". Joint Research Report R94:03 and CRC-043, SICS and SRI International, Kista and Cambridge, January 1994.

[Alshawi *et al* 1991]
Alshawi, H., D. Carter, B. Gambäck, M. Rayner (1991). "Translation by Quasy logical Form Transfer" in *Proceedings of the 29th Annual Meeting of the Association of Computational Linguistics*, Berkeley, pp. 161- 167.

[Alshawi (ed.) 1992]
Alshawi, H. (ed.) (1992). *The Core Language Engine*, Cambridge, Massachusetts: The MIT Press.

[Andersson *et al* 1993]
Andersson, J., S. Andersson, K. Boortz, M. Carlsson, H. Nilsson, T. Sjöland, J Widén (1993). "SICStus Prolog User's Manual Version 2.1 #8", SICS Technical Report - T93:01, Kista.

[Ejerhed *et al* 1992]
Ejerhed, E., G. Källgren, O. Wennstedt, M. Åström (1992). *The Linguistic Annotation System of the Stockholm-Umeå Corpus Project*, Department of General Linguistics, University of Umeå, Report no. 33, Umeå.

[Gambäck 1992]
Gambäck, B. (1992). "Lexical Acquisition: the Swedish VEX System", SICS Research Report R92:12, Kista.

[Gambäck *et al* 1994]

Gambäck, B., J. Karlgren, C. Samuelsson (1994). "Natural Language Interpretation in Prolog", SICS Perspective Report, Kista.

[Gambäck & Ljung 1993]
Gambäck, B. and S. Ljung (1993). "Question Answering in the Swedish Core Language Engine" in *Scandinavian Conference on Artificial Intelligence -93*, E. Sandevall & C.G. Jansson (ed.) IOS Press, Amsterdam, pp. 212-225.

[Gambäck & Rayner 1992]
Gambäck, B. and M. Rayner (1992). "The Swedish Core Language Engine", 3rd NOTEX, Linköping 1993, pp. 71-85.

[Garside *et al* 1987]
Garside, R., G. Leech, G. Sampson (1987). *The Computational Analysis of English*, Essex: Longman.

[Källgren, 1990]
Källgren, G. (1990). "'The first million is hardest to get': Building a Large Tagged Corpus as Automatically as Possible", Proceedings of Coling '90, Helsinki, Vol. 3, pp. 400-404.

[Källgren, 1991]
Källgren, G. (1991). "Storskaligt korpusarbete på dator: En presentation av SUC-korpusen", Svenskans beskrivning 18, Lund.

[Källgren, 1993]
Källgren, G. (1993). "The Stockholm - Umeå Corpus Project: Corpus-Based Research on Models for Processing Unrestricted Swedish Text", Presentation written for the Evaluation of the NUTEK/HSFR Language Technology Research Program, Stockholm.

[Marcus *et al* 1993]
Marcus, M. P., B. Santorini and M. A. Marcinkiewicz (1993). "Building a Large Annotated Corpus of English: The Penn Treebank", Computational Linguistics Vol 19, Number 2,

pp. 313-330.

[Sampson 1992]
This information was found in an Internet announcement, 6th September 1992. In this document, a forthcomming book, *English for the Computer*, by Geoffry Sampson, Oxford University Press, is mentioned.

[Teleman 1974]
Teleman, U. (1974). *Manual för Grammatisk Beskrivning av Talad och Skriven Svenska*, Lundastudier i Nordisk Språkvetenskap, serie C, Nr 6, Studentlitteratur, Lund.

Appendix A

Sample lexicon (*second_lexicon* strategy) based on SUC tags for the sentence *Hur genomför man ämnesbyte?*. The verb gets several entries, to cover the most frequent types of verb.

```
lex(hur,@positional_adverb).

lex('genomf|r',
  [v:
    [lexform='genomf|ra',synmorphv=irreg,nullmorphv=n,
     vform=present/\fin,passive=n,gap=n,@gap_island,
    subcat=[]]]).

lex('genomf|r',
  [v:
    [lexform='genomf|ra',synmorphv=irreg,nullmorphv=n,
     vform=present/\fin,passive=n,gap=n,gaps=A,
     subcat=[@np_object(A,n,n)]]]).

lex('genomf|r',
  [v:
    [lexform='genomf|ra',synmorphv=irreg,nullmorphv=n,
     vform=present/\fin,passive=n,gap=n,gaps=A,
     subcat=[@np_sentential(A,B,C,norm\/q)]]]).

lex('genomf|r',
  [v:
    [@gap_thread(A,B),lexform='genomf|ra',synmorphv=irreg,
     nullmorphv=n,vform=present/\fin,passive=n,gap=n,
     subcat=[@np_object(A,n,C),@np_sentential(B,C,D,norm\/q)]]
  ]).

lex('genomf|r',
  [v:
    [lexform='genomf|ra',synmorphv=irreg,nullmorphv=n,
     vform=present/\fin,passive=n,gap=n,gaps=A,
     @agr(B),subcat=[@vp_event(inf\/att,A,B,C,D)]]]).
```

```
lex('genomf|r',
   [v:
     [@gap_thread(A,B),lexform='genomf|ra',synmorphv=irreg,
      nullmorphv=n,vform=present/\fin,passive=n,gap=n,
    @agr(C),
      subcat=[@np_object(A,n,D),@vp_event(inf\/att,B,C,E,F)]
    ]]).

lex(man,@det_indef_pronoun([y,y,A],B)).

lex('{mnesbyte',
   [nbar:
     [common=n,@takes_infix(n),simple=y,
      @gap_island,subcat=[],def=n\/poss,@relagr(B),
      @agr(B),sing=y,person=3,conjoined=n,genitive=n,
      synmorphn=irreg,measure=C,lexform='{mnesbyte']]).
```

Appendix B

Samples of CLE analyses. In the first example, only one analysis was produced:

**** 2 ** hur genomf|r man {mnesbyte *******

There is 1 analysis.

```
[sigma_whq_Movement-1,
 [[s_pp_s_WhMvt-2,
  [[lex-3,hur],
   [s_v_np_vp_Inv-4,
    [[lex-5,genomf|r],[lex-6,man],
     [vp_vp_pp_OptMod-7,
      [[vp_v_comp_Normal-8,
        [[v_Gap-9,[]],[np_nbar_Mass-10,[[lex-11,{mnesbyte]]]
        ]],
       [pp_Gap-12,[]]]]]]]]]
```

In the next example, the parser found the sentence ambiguous. Two analyses were given. The alternative reading is marked "/|\", and has to do with the interpretation of *detta*, which might make up an elliptic NP on its own (the **np -> det + NbarEllipsis** rule).

**** 3 ** man kan g|ra detta smidigt och n{stan om{rkligt eller abrupt *******
There are 2 analyses.

```
[sigma_decl-1,
 [[s_np_vp_Normal-2,
  [[lex-3,man],
   [vp_v_comp_Normal-4,
    [[lex-5,kan],
     [vp_vp_advp_OptMod-6,
      [/|\([vp_v_comp_Normal-7,
          [[lex-8,[g|r,-a]],[lex-9,detta]]],
         [vp_v_comp_Normal-10,
```

```
    [[lex-8,[g|r,-a]],
     [np_det_NbarEllipsis-11,[[lex-12,detta]]]]),
[advp_advp_conj_advp-13,
 [[advp_advp_conj_advp-14,
   [[lex-15,smidigt],[lex-16,och],
    [advp_deg_advp-17,
     [[lex-18,n{stan],[lex-19,om{rkligt]]]]],
  [lex-20,eller],[lex-21,abrupt]]]]]]]]]]
```

An example of a sentence for which the parsing did not terminate within the space of the time-out:

**** 9 ** utan att beh|va f|rklara eller be om urs{kt b|rjar vi omedelbart att spekulera i -- och titta efter -- om n}gon p} gatan tog skada *******

!!! TIME OUT, BABY, after 1200 seconds !!!

For this sentence, the parser did not find any sigma rule, only partial parses were found:

**** 14 ** pl|tsligt s{ger mormor : titta en s} stor mask han f}r upp *******

No parses found.

!!! NO PARSES FOUND (baby?!) !!!

Shortest sequence(s) of words and complete constituents:

pl|tsligt s{ger mormor: titta NP(?!):4 han f}r upp